Loading the dictionary, part 4: Character conversion redux

devblogs.microsoft.com/oldnewthing/20050516-30

May 16, 2005

Raymond Chen

Getting rid of `getline` was a big help, but 480ms is still not quite peppy enough. You need to respond to user actions within a tenth of a second for thing to seem responsive.

Profiling the latest endeavor reveals that 40% of our CPU time is spent in `codecvt::in`. Some debugging reveals that `codecvt::in` ultimately calls `MultiByteToWideChar` but uses it to convert only one or two characters at a time, even though we handed it a whole line.

Let's get rid of `codecvt::in` and convert the characters ourselves, calling `MultiByteToWideChar` exactly once to convert the entire line at a single go.

```
#define CP_BIG5 950
Dictionary::Dictionary()
{
 MappedTextFile mtf(TEXT("cedict.b5"));
 // typedef std::codecvt<wchar_t, char, mbstate_t> widecvt;
 // std::locale l(".950");
 // const widecvt& cvt = _USE(l, widecvt); // use_facet<widecvt>(l);
 const CHAR* pchBuf = mtf.Buffer();
 const CHAR* pchEnd = pchBuf + mtf.Length();
 while (pchBuf < pchEnd) {
  const CHAR* pchEOL = std::find(pchBuf, pchEnd, '\n');
  if (*pchBuf != '#') {
   size_t cchBuf = pchEOL - pchBuf;
   wchar_t* buf = new wchar_t[cchBuf];
   DWORD cchResult = MultiByteToWideChar(CP_BIG5, 0,
                           pchBuf, cchBuf, buf, cchBuf);
   if (cchResult) {
    wstring line(buf, cchResult);
    DictionaryEntry de;
    if (de.Parse(line)) {
     v.push_back(de);
    }
   }
   delete[] buf;
  }
  pchBuf = pchEOL + 1;
 }
}
```

Instead of using the `codecvt::in` method to perform character conversion, we go straight to the `MultiByteToWideChar` function. Notice that we assume that the Big5 string will not generate more Unicode characters than its length in bytes. This happens to be a safe assumption based on our external knowledge of the Big5 encoding. (If the encoding were something else, the assumption may no longer be valid.)

With this change, the dictionary load time has dropped to 240ms (or 300ms if you include the time it takes to destroy the dictionary). That's twice as fast the previous version, but still not quite close enough to the 100ms goal. We still have some work ahead of us.

[Raymond is currently on vacation; this message was pre-recorded.]

Raymond Chen

Follow